Auction Algorithm for Production Models

Junghwan Shin *and Sanjiv Kapoor †

2011

Abstract

We show an auction-based algorithm to compute market equilibrium prices in a production model, where consumers purchase items under separable nonlinear utility concave functions which satisfy W.G.S(*Weak Gross Substitutes*); producers produce items with multiple linear production constraints.

Our algorithm differs from previous approaches in that the prices are allowed to both increase and decrease to handle changes in the production. This provides a tâtonnement style algorithm which converges and provides a PTAS. The algorithm can also be extended to arbitrary convex production regions and the Arrow-Debreu model. The convergence is dependent on the behavior of the marginal utility of the concave function.

1 Introduction

The market equilibrium is a well studied problem for a general market model which includes production constraints[2]. Arrow and Debreu[1] had introduced a general production model for exchange markets and have shown proof of existence of equilibria. In the Arrow-Debreu model, each production schedule lies in a specified convex set. When the production model is constrained by positive production vectors only, convex programs have been obtained[28][26] for linear cases. There has been considerable recent research on the complexity of computing equilibria [4, 10, 11, 12, 15, 17, 21].

To solve the market equilibrium problem, a number of approaches have been used including convex programming, auction-based algorithm[17] and primal-dual[10] methods.

Two techniques, the primal-dual schema and auction-based algorithms, have mainly been successful for models satisfying W.G.S.(weak gross substitutability). Other technique includes tâtonnement processes.

Jain et al.[23] further generalize to a model with homothetic, quasi-concave utilities which is introduced in [14] and [15]. However, the paper is restricted in that the concave functions are assumed to be homogeneous of degree one. We differ in that we consider functions that are *Weak Gross Substitutes*. Jain et al.[22] also give an explicit, polynomial sized convex program for the production planning model with linear utilities. These papers utilize convex programming. Codenotti et al. [9] consider gross-substitute functions with positive production constraints and provide approximation results via the ellipsoid method. Recent results that include production within the model include the work on price discrimination model [20]. Our results apply to convex production sets that are not constrained to be positive.

For the production model when there is one constraint an auction-based algorithm is provided in [25]. In this paper, we give an auction-based algorithm for a production model in which consumers have separable utilities for items that satisfy the weak gross substitutes property. Furthermore, producers have multiple linear production constraints. Note that we can also consider producers that sell items to maximize their profit as well as purchae materials or resources to produce them. In this model, note that each buyer chooses a subset of items that maximizes her utility and each producer chooses a feasible production plan that maximizes his profit at current prices.

While auction methods have been applied to market equilibrium problems, the key aspect of many algorithms ([19][25]) is that price discovery is monotone since goods are always sold out, i.e. overdemanded. In this model, the change in production plans during the course of the algorithm induces oversupply of goods. This implies that price discovery cannot be monotone. This is a critical difference from previous methods. Thus unlike previous auction-based algorithms for consumer and production models, we consider

*Computer Science Ph.d. Candidate, IIT, Chicago, IL, e-mail: jshin7@iit.edu
†Computer Science, IIT, Chicago, IL, e-mail: kapoor@iit.edu

auction algorithm which decrease price also. Since each producer has an arbitrary number of production constraints, updating prices affects production schedule. When producers chooses a bundle of items which also maximize consumer utilities, decreasing the prices is not required. However, when profitable items are not demanded by consumers, price decrease may be needed. Bounding the decrease in price can be done by small increments of the production plan along a gradient direction specifying increasing profits. The direction is obtained from the convex program describing the production plans. The step length is dictated by the behavior of the utility functions, in particular on the behavior of the marginal utility function. For simplicity of presentation our results are shown in the Fisher model with linear production constraints. They can be extended to arbitrary convex production regions and the Arrow-Debreu model.

This paper is organized as follow: In Section 2, we define the production market model and invariance of the algorithm to show the correctness and the convergence. We describe the overall idea on the auction-based algorithm and describe in market states and market state transitions in Section 3. Further, procedures are described in Section 4 and we show invariance conditions that ensure optimality in Section 5. In Section 6, we finally show the correctness of the algorithm in the algorithm and evaluate the complexity. In the Appendix, we will describe the details of the algorithm.

2 Production Model

We consider a market equilibrium problem with production, termed as *MEP*, with *nonlinear utility functions* and *multiple linear production constraints*. In here, we consider nonlinear utility functions satisfy W.G.S. property. The production model we consider has q producers, along with n consumers(traders) and m items. For simplicity, we assume that every item is produced in a quantity greater than or equal to ϵ.

Consumer i has a utility function, termed as $U_i(X_i) = \sum_j u_{ij}(x_{ij})$, and fixed initial endowment, e_i. x_{ij} represents the amount of allocation on item j to consumer i, $X_i = (x_{i1}, x_{i2}, ..., x_{im})$ denotes the current allocation vector on the items to consumer i, and $u_{ij}(x_{ij}) : R^+ \to R^+$ is the function representing the utility of item j to consumer i. Our assumption is that u_{ij} is separable. We denote by v_{ij} the first derivative of U_i w.r.t. x_{ij}.

We let P be the vector of prices of the items, where the j-th component p_j represents the price of item j. Then, we can represent *bang-per-buck*, $\forall i, j : \alpha_{ij} = v_{ij}(x_{ij})/p_j$, and let α_i be *max bang-per-buck* s.t. $\alpha_i = \max_j \alpha_{ij}$.

For producers, let z_{sj} represent the quantity on item j produced by producer s and sold or bought by producer s. Also, there are non-manufactured raw items defined as a_j for all items. Producer s gains profit $\sum_j p_j z_{sj}$ when all items are sold out at the price P. We assume that the production schedule is constrained by a set of linear inequalities. Suppose producer s has l_s linear constraints. Then, $\forall s, \ell : \sum_j z_{sj} a^{\ell}_{sj} \le K^{\ell}_s$, where a^{ℓ}_{sj} and K^{ℓ}_s are constants determined by the production schedules by producer s. $\forall s, j : z_{sj} \in \mathbb{Z}$.

Given a fixed vector of prices P, the following linear programs, $CP_i(P)$ and $PP_s(P)$ represent the optimal consumption and production schedule, respectively.

Consumer nonlinear programming for consumer i

$$CP_i(P)\textbf{:}\quad \text{Maximize } \sum_{j=1}^{m} U_i(X_i) \text{ subject to}$$

$$\sum_{j=1}^{m} x_{ij} p_j \le e_i \quad \forall i \tag{1}$$

$$x_{ij} \ge 0 \qquad \forall i, j \tag{2}$$

Production linear programming for producer s

$$PP_s(P)\textbf{:}\quad \text{Maximize } \sum_{j=1}^{m} p_j z_{sj} \text{ subject to}$$

$$\sum_j z_{sj} a^{\ell}_{sj} \le K^{\ell}_s \quad \forall s, \ell \tag{3}$$

$$z_{sj} \ge 0 \qquad \forall s, j \tag{4}$$

Furthermore, we assume that utilities for the items satisfy the *Weak Gross Substitute* property. Items are said to be weak gross substitutes for a buyer iff increasing the price of any item does not decrease the buyer's demand for other items. Similarly, items in an economy are said to be weak gross substitutes iff increasing the price of any item does not decrease the total demand of other items.

Consider the consumer maximization problem $CP_i(P)$. Let $S_i(P) \subset R_+^m$ be the set of optimal solutions of the program $CP_i(P)$. Consider another price vector $P' > P$. Items are gross substitutes for buyer i if and only if for all $X_i \in S_i(P)$ there exists $X'_i \in S_i(P')$ such that $p_j = p'_j \Rightarrow x_{ij} \le x'_{ij}$.

Note that since u_{ij} is concave, v_{ij} is a non-increasing function. The following result in [17] characterizes the class of separable concave gross substitutes utility functions.

Lemma 2.1 *[17] Items are gross substitutes for buyer i if and only if for all j, $yv_{ij}(y)$ is a non-decreasing function of the scalar y.*

ϵ-equilibrium in the Production Model We define an ϵ-equilibrium in the production model as follows:

1. *Sold-out condition:* All produced items are sold to consumers within a factor of $(1+\epsilon)$.
2. *Bpb condition:* The bang per buck on the items purchased by consumer i should be almost the same.
3. *Opt-prod condition:* The current production plan almost maximizes the profit of each producer.
4. *Termination condition:* Consumers spend all their endowment within a factor of $(1+\epsilon)$.

The first three condition are referred to as conditions OPT and are detailed below:

$$I_1 : \forall j : \sum_s z_{sj}/(1+\epsilon) \le \sum_i x_{ij} \le \sum_s z_{sj} \tag{5}$$

$$I_2 : \forall i,j : x_{ij} > 0 \Rightarrow v_{ij}(x_{ij}) \le \alpha_{ij} p_j \le (1+\epsilon) v_{ij}(x_{ij}) \tag{6}$$

$$I_3 : \forall s,j : p_j \hat{z}_{sj} \le (1+\epsilon) p_j z_{sj} \tag{7}$$

where $\hat{z}_{sj}$ is the optimal amount of item j produced by s. Equilibrium is established together with the termination condition below:

$$I_4 : \forall i : r_i \le \epsilon e_i \tag{8}$$

3 Auction Algorithm for the Production model

3.1 Notations and Preliminaries

We define the *demand set* of consumer i as $D_i = \{j : v_{ij}(x_{ij})/((1+\epsilon)p_j) \le \alpha_i \le v_{ij}(x_{ij})/p_j\}$. We work with a discretized price space, and at any instant a good is sold at at two level prices: p_j and $p_j/(1+\epsilon)$. We let h_{ij} represent the amount of item j that consumer i buys at price p_j, and y_{ij} represent the amount of item j that consumer i purchases at price $p_j/(1+\epsilon)$. x_{ij} which is the summation of h_{ij} and y_{ij} represents the total quantity of allocation on item j by consumer i. We also let r_i denote *residual money* which can be calculated as $e_i - \sum_j (p_j h_{ij} + p_j y_{ij}/(1+\epsilon))$.

We denote by $\mathcal{Z}_s$ a set of *feasible production plan(schedule)* of producer s i.e., $\mathcal{Z}_s = \{Z : A^T Z \le K_s\}$, where A represents a $m \times \ell$ matrix whose row and column correspond to items and constraints, respectively. Z represents a production plan and K_s represents capacity constraints, $Z, K_s \in R^m$. Note that Z is an unconstrained vector, that is, if a producer produces item j as a product, then Z may be positive; if a producer consumes item j as a material or resource, then Z may be negative. Let us define a *profitable* production plan of producer s as $\hat{Z}_s = \arg\max_{Z \in \mathcal{Z}_s} \{P^T Z | A^T Z \le K_s\}$ at price P.

We define a set of *over-demanded* items $\mathcal{O}_d = \{j : \exists i \text{ s.t. } \sum_s z_{sj} < \sum_i x_{ij}\}$. Moreover, we define a set of *over-supplied* items as $\mathcal{O}_s = \{j : \sum_s z_{sj} > \sum_i x_{ij}\}$.

3.2 Algorithm Overview

We apply the auction mechanism to solve the problem of finding equilibrium in the linear production model. Note that in this application we allow the algorithm to decrease and increase prices. The market equilibrium in the production model is to find a price vector at which consumers maximize their utility and producers maximize their profit. Consumers want to purchase items at lower price, while producers want to sell items at higher price, creating complementary optimization constraints. In fact, the producers would wish to produce as much demand as the market can sustain, to maximize their profit. The algorithm we design allows producers to increase supply incrementally to change their production plan according to the current market price.

The algorithm is iterative and its progress can be measured by the changes in its state during successive steps. The algorithm we design allows producers to increase supply incrementally to change their production plan according to the current market price. To prove convergence we will show that the producer's profit increases during phases of the algorithm.

Overview of the algorithm The algorithm starts with an initialization , procedure *initialize* where the price vector is initialized, each producer is assigned an initial production schedule and each consumer is assigned goods, all quantities small enough. In procedure *algorithm_main*, the algorithm determines if consumers have no extra demand and producers have no way to increase their profit and consequently stops. The invariant conditions to be maintained at the end of every phase are conditions (5),(6) and (7). If at the beginning of a phase no consumer has any residual money left, the procedure can terminate satisfying all the conditions. Otherwise, it is determined if the consumer can outbid other consumers to acquire her desired items in procedure *satisfy_demand* and subsequently adjust the allocations of other consumers *adjust_bpb* to ensure condition (6). Each of these procedures may result in an increase in price. Consequently, the production schedule may required to be changed. This is done in procedure *prod_reschedule*.

Algorithm 3.1 algorithm_main

initialize
          // bpb, opt-prod and sold-out conditions hold.
while there exists consumer i with residual money **do**
    update bang-per-buck
          // bpb, opt-prod and sold-out conditions hold.
    satisfy_demand(i)
          // sold-out condition holds.
    adjust_bpb
          // bpb and sold-out conditions hold.
    prod_reschedule
          // bpb, opt-prod and sold-out conditions hold.
end while
          // termination, bpb, opt-prod and sold-out conditions hold.

Algorithm 3.2 satisfy_demand(i)

consumer i with extra demand purchase items

update a set of demanded items of consumer i
if item j is available at lower price from consumer k **then**
    outbid(i, k, j, α_{ij})
    update bang-per-buck
else
    raise_price(j)
end if

The procedure, *prod_reschedule*, determines a feasible direction that improves the profit of a producer, by using a linear program.A small step of length δ is taken along the feasible direction. The step length is chosen so that optimality conditions hold as shown later. The step provides a new production plan based on the current price vector P, and the production of items may decrease and increase as compared with

Algorithm 3.3 adjust_bpb

bang-per-buck condition holds by outbid

while bang-per-buck not hold, i.e $\exists i : r_i > 0$ and $\exists j : \alpha_{ij} p_j < v_{ij}(x_{ij})$ **do**
  if item j is available at lower price from consumer k **then**
    outbid(i, k, j, α_{ij})
  else
    raise_price(j)
  end if
end while

the previous production plan. Also, note that z_{sj} may be negative in that producer s consumes item j to produce other items; The change in the production of items leads to goods that are either oversupplied or over demanded. The demand and supply are then balanced in the procedures: *bal_od* and bal_os.

Due to a change in the production plans, a good j may be reduced in quantity. Then the consumers holding the good h have reduced allocation and have money m, returned to them. To ensure their bang-per-buck conditions, consumers outbid other consumers in procedure *bal_od*. After procedure *bal_od*, procedure *adjust_bpb* is called to ensure that bang-per-buck conditions are met. Note that this may involve a rise in price.

A change in production may also involve increased production of some particular good. In procedure *bal_os*, the algorithm will balance over-supplied items so that there is no over-supplied item in the market.

Procedure *bal_os* balances the demand and supply of over-supplied items by four different procedures : *purchase_money*, *transfer_money*, *sell_lprice* and *decrease_price*. Procedure *purchase_money* is allowed for consumer with extra demand to acquire items. Unless consumer has residual money, procedure *transfer_money* will take money from other items to purchase her demand items. If the market has an item that is over-supplied, then producers will provide items at a lower price as shown in procedure *sell_lprice*. Finally, Procedure *decrease_price* will be called if there is no available item at a higher price.

Algorithm 3.4 prod_reschedule

while execute *lp_solver* and if not at optimality, $\forall s, j : z'_{sj} := z_{sj} \pm \epsilon' \sum_s z_{sj}/q$ **do**
  if producer can increase profit **then**
    bal_od$(j) : \forall j \in \mathcal{O}_d$
    adjust_bpb
        // bpb condition holds.
    bal_os $: \forall j \in \mathcal{O}_s$
        // bpb and sold-out conditions hold.
    check_profit
  else
    break
  end if
end while
    // bpb, sold-out and opt-prod condition hold.

Finally, the algorithm checks whether there is total profit increase in the overall process at each iteration. Procedure *check_profit* will be called to check whether consumers spend more than they did at the prior iteration.

If the total profit does not increase, then changes of productions plans are voided. It is shown later that in this case the production plans are at near optimality. The algorithms then exits *prod_reschedule*.

Choosing a value of ϵ' We will assume that the producers will change their production plan by a factor of $1 + \epsilon'$. We choose ϵ' to ensure an approximation of $(1 + \epsilon$.

We choose constants as follows: set ϵ_1 s.t. $\forall i, j : v_{ij}(x_{ij})/(1 + \epsilon) \leq v_{ij}((1 + \epsilon_1)x_{ij}) \leq v_{ij}(x_{ij})$ and $\forall i, j : v_{ij}(x_{ij}) \leq v_{ij}((1 - \epsilon_1)x_{ij}) \leq (1 + \epsilon)v_{ij}(x_{ij})$. Let $\epsilon_2 = \epsilon^3/e, e = \sum_i e_i$ and let $\epsilon' = \min(\epsilon_1, \epsilon_2)$.

3.3 States of the market

The state of the market at time instant t is represented as a 3-tuple $(P(t), Z(t), X(t))$, where $P(t), Z(t)$ and $X(t)$ represent the price vector, the production plan and the current allocation (or demand) at time t, respectively. We will also use $p_j(t), z_{sj}(t)$ and $x_{ij}(t)$ to denote the price of item j, the production plan for item j by producer s and the demand of consumer i for item j at time t, respectively.

We consider the transitions of the state of the market at time t, $(P(t), Z(t), X(t))$, to the states at time $t+1$, $(P(t+1), Z(t+1), X(t+1))$ via an algorithmic process.

State Transitions of the market We consider the state transitions :

$$P(t+1) = f_P(P(t), Z(t), X(t)) \tag{9}$$
$$X(t+1) = f_X(P(t), Z(t), X(t)) \tag{10}$$
$$Z(t+1) = f_Z(P(t), Z(t), X(t)) \tag{11}$$

where we design the functions f_P, f_X and f_Z to satisfy conditions (5) through (8)on the state of the market. The state transition function f_P, f_X and f_Z are computed in algorithm_main. At each iteration conditions (5) through (7) should be satisfied. Furthermore we will show that at termination, condition (8) is true and thus at termination the market is in a state of equilibrium.

The function f_P depends on a parameter $P(t), Z(t)$ and $X(t)$. In other words, $p_j(t)$ will increase by a factor of $(1+\epsilon)$ unless $\exists i : y_{ij}(t) > 0$. The procedure outbid may occur on either consumer's side or producer's side. Consumer i outbids consumer k to acquire item j, or producer s outbids consumer k to consume item j to produce item j'.

To compute f_X we invoke procedure satisfy_demand which allows consumer i with $r_i > \epsilon e_i$ to purchase item $j \in D_i$ at $p_j(t)$ by acquiring item j from consumer k. If the demand and the supply does not match, then procedures outbid, bal_os or bal_od may be called so that $X(t+1)$ may change.

To update the production schedule and compute f_Z, we invoke procedure prod_reschedule. According to $P(t)$, a linear program solver returns an optimal production schedule. As mentioned before, a next production schedule is determined by the demand and the supply.

3.4 Market State Transitions

States of market in procedure satisfy_demand We now detail the change of the states of market as well as maintenance of invariants in procedure satisfy_demand inside procedure algorithm_main. In a market state, $(P(t), Z(t), X(t))$, procedure satisfy_demand calls either outbid or raise_price when $\forall j : \alpha_{ij} p_j = v_{ij}(x_{ij})$. Since there exists consumer i s.t. $r_i > \epsilon e_i$, she will buy item $k = \arg\max_j \alpha_{ij}$.

Note that before procedure satisfy_demand, $\forall j : \alpha_{ij} = v_{ij}(x_{ij}(t))/p_j(t)$. Then, for item $j \neq k$, $\alpha_{ij} p_j(t) = v_{ij}(x_{ij}(t))$ and $\alpha_{ij} p_j(t+1) = v_{ij}(x_{ij}(t+1))$. In the case of item k, $\alpha_{ik} p_k(t) = v_{ik}(x_{ik}(t))$ and $\alpha_{ik} p_k(t+1) = (1+\epsilon) v_{ik}(x_{ik}(t+1))$. However, α_{ik} will be updated, and finally, $\forall j : \alpha_{ij} p_j(t+1) = v_{ij}(x_{ij}(t+1))$.

Since there is, $P(t)$ and $X(t)$ may be not equal $P(t+1)$ and $X(t+1)$, respectively. However, no change in production plan, $Z(t) = Z(t+1)$.

States of market in procedure adjust_bpb We next consider changes to the states of the market and the invariants in procedure adjust_bpb inside procedure algorithm_main. In a market state, $(P(t), Z(t), X(t))$, procedure adjust_bpb calls either outbid or raise_price when $\exists j : \alpha_{ij} p_j < v_{ij}(x_{ij})$. For the corresponding consumer i, $\forall j : \alpha_{ij} p_j = v_{ij}(x_{ij})$ at the end of adjust_bpb.

For item j s.t. $\alpha_{ij} p_j(t) < v_{ij}(x_{ij}(t))$, procedure outbid makes $p_j(t) = p_j(t+1), x_{ij}(t) < x_{ij}(t+1)$; on the other hand, procedure raise_price makes $(1+\epsilon) p_j(t) = p_j(t+1)$. The aim of this procedure is $\alpha_{ij} p_j(t+1) = v_{ij}(x_{ij}(t+1))$.

Therefore, $P(t)$ and $X(t)$ may be not equal to $P(t+1)$ and $X(t+1)$, respectively. Since there is no change in production plan, $Z(t) = Z(t+1)$.

States of market in procedure prod_reschedule Given $(P(t), Z(t), X(t))$ (the current market state) procedure prod_reschedule calls lp_solver which returns optimal production plans, $\hat{Z}_s$ according to $P(t)$ (as shown in line 2 in procedure prod_reschedule). $Z_s(t+1)$ depends on $Z_s(t)$ and $\hat{Z}_s$ as follows:

$$\forall s, j : z_{sj}(t+1) = \begin{cases} z_{sj}(t) + \Delta,\ \Delta = \epsilon' \sum_s z_{sj}(t)/q & \text{if } \|\Delta\| \leq ||\hat{z}_{sj}(t) - z_{sj}(t)|| \\ \hat{z}_{sj}(t) & \text{otherwise} \end{cases}$$

Since the production plan changes, allocations of consumers and production of producers might change. These changes may increase or decrease price of items. We describe more details in Section 4.

4 Other procedures in the algorithm

In this section we will discuss other important and their properties.

1. outbid(i, j, k, α) : consumer i with surplus will outbid consumer k to acquire item j. The quantity that is outbid of item j is determined by the utility function and the current allocation so as to maintain the invariance. After we set the amount outbidden, we need to update the allocation and the surplus of both consumers i and k.

2. purchase_money$(i, j, t_{oversupply})$: for item $j \in \mathcal{O}_s$, the procedure will check whether item j is demanded by consumer i with surplus. If exists, consumer i will purchase item j as much as the minimum of the quantity oversupplied and extra demand. Here we also need to update the allocation of consumer i on item j and the surplus.

3. transfer_money$(i, j, t_{oversupply})$: in the case when there is no consumer with surplus for item $j \in \mathcal{O}_s$, the procedure transfer_money will be called. That is, consumer i with demand on item j will give up item j' and spend the returning money to buy item j. The allocation of consumer i will be updated for item j and j'.

4. sell_lprice$(i, j, t_{oversupply})$: if there is consumer neither with surplus nor with items transferable, producers will offer item j at lower price. Remember that the algorithm accepts two-level price: a higher price and a lower price by a factor of $(1 + \epsilon)$. Producers are forced to supply item j at a lower price, then either j is fulfilled or no consumer has item j at a higher price.

5. decrease_price(j) : when procedure sell_lprice fails to fulfill item j, we assure that the allocation of item j is at a lower price to any consumer. Then, procedure decrease_price will be called, and the while statement of procedure bal_os will check whether procedures purchase_moeny, transfer_money or sell_lprice might be called.

6. raise_price(j) : after the procedure prod_reschedule, if there is a consumer with surplus then consumer i outbids other consumers. If consumer i is not satisfied then she will increase the price of item j by a factor of $(1+\epsilon)$. Procedure raise_price itself does not change the allocation, but affects invariance that should be maintained in the algorithm. As consumers acquire item j at higher price than previous, the producers may change their production plan. This will be checked in the program.

5 Invariances

In this section we will prove the invariances that hold during the course of the algorithm. We first discuss the properties of two procedures.

5.1 Procedure raise_price

We consider the number of occurrences of procedure raise_price.

Lemma 5.1 *In procedure prod_reschedule at most one occurrence of procedure raise_price is required.*

Proof: We claim that procedure raise_price occurs at most once per each item at each iteration in procedure prod_reschedule.

Definitely, procedure bal_os will not call procedure raise_price. Procedure prod_outbid allows producers to outbid consumers , but procedure raise_price will not be called.

Procedure bal_od will be called , and consumer i may potentially violate condition (6) because $v_{ij}(x_{ij})$ increases when x_{ij} decreases. To adjust bang-per-buck condition, procedure adjust_bpb will be called.

If $\forall k : y_{kj} = 0$ and $\alpha_{ij} p_j < v_{ij}(x_{ij})$, then procedure raise_price will be called once. Note that $\alpha_{ij} p_j = v_{ij}(x_{ij}) \le v_{ij}(x'_{ij}) \le (1+\epsilon)\alpha_{ij} p_j$, where x'_{ij} corresponds to the reduced amount of item j of consumer i. Since ϵ' is chosen to satisfy $v_{ij}(x_{ij}) \le v_{ij}((1-\epsilon')x_{ij}) \le (1+\epsilon) v_{ij}(x_{ij})$, $\alpha_{ij} p_j \le v_{ij}(x'_{ij}) \le (1+\epsilon)\alpha_{ij} p_j$. After raise_price, i.e. $p'_j = (1+\epsilon)p_j, \alpha_{ij} p'_j \ge v_{ij}(x'_{ij})$.

Therefore, at most one occurrence of procedure raise_price will be enough in procedure prod_reschedule. □

5.2 Procedure decrease_price

Similarly, we consider the number of occurrences of procedure decrease_price.

Lemma 5.2 *Procedure decrease_price of each item occurs at most once per each iteration in procedure prod_reschedule.*

Proof: Let p and p' denote a previous price vector and a current price vector, respectively. Let z and z' denote a previous production plan and a current production plan, respectively.

Suppose, for contradiction, that procedure decrease_price on item j occurs at least twice in an iteration. Remember that we have two additional variables ϵ_1 and ϵ_2 and $\epsilon' = \min(\epsilon_1, \epsilon_2)$ in a paragraph in Section 3. It is enough to show the case when $\epsilon' = \epsilon_2 = \epsilon^3 / \sum_i e_i$.

The money required to consume the oversupplied items is

$$\epsilon' \sum_s \sum_{j'} z_{sj'} p_{j'} = \epsilon^3 \sum_s \sum_{j'} z_{sj'} p_{j'} / \sum_i e_i$$

Calling decrease_price on item j returns the following amount of money to all consumers,

$$((\epsilon^2 + 2\epsilon) \sum_i h_{ij} + \epsilon \sum_i y_{ij}) p_j$$

For decrease_price to occur more than once, the required money must be greater than the money returned, i.e.

$$\frac{\epsilon^3 \sum_s \sum_{j'} z_{sj'} p_{j'}}{\sum_i e_i} > ((\epsilon^2 + 2\epsilon) \sum_i h_{ij} + \epsilon \sum_i y_{ij}) p_j$$

Since $\sum_s \sum_{j'} z_{sj'} p_{j'} \le \sum_i e_i$,

$$\begin{aligned}
&\epsilon^3 > ((\epsilon^2 + 2\epsilon) \sum_i h_{ij} + \epsilon \sum_i y_{ij}) p_j \\
&\Rightarrow \epsilon^2 > ((\epsilon + 2) \sum_i h_{ij} + \sum_i y_{ij}) p_j > \sum_i x_{ij} p_j \\
&\Rightarrow \epsilon^2 > \sum_i x_{ij} p_j \ge \epsilon^2
\end{aligned}$$

Since every item is demanded by at least one consumer, the money spent on item j , $\sum_i x_{ij} p_j$ should be equal or at least ϵ^2, i.e. $\sum_i x_{ij} p_j \ge \epsilon^2$. This is a contradiction. □

Ensuring sold-out condition (5): Procedures adjust_bpb and satisfy_demand will not violate the invariance because there is no change in production plan and the amount of items sold does not decrease.

When producer s changes a production plan from $Z_s(t)$ to $Z_s(t+1)$, the following cases arise.

- $\sum_s z_{sj}(t+1) - \sum_i x_{ij}(t+1) > 0$
- $\sum_s z_{sj}(t+1) - \sum_i x_{ij}(t+1) < 0$

Procedure bal_od resolves the first case, and the second case can be resolved in procedure bal_os. That is, procedure bal_od balances all items in $\mathcal{O}_d$; procedure bal_os balances oversupplied items.

1. For $j \in \mathcal{O}_d$, if $\exists i : x_{ij} > 0$ then $x_{ij}(t+1) = x_{ij}(t) - \min(x_{ij}(t), \xi)$ shown in procedure bal_od.

2. For $j \in \mathcal{O}_s$, if $\exists i : j \in D_i$ and $r_i > \epsilon e_i$, then consumer i will purchase item j by using procedure purchase_money (as shown in line 2 and 3 in procedure bal_os). It ensures that $p_j(t) = p_j(t+1)$ and while $\sum_s z_{sj}(t+1) = \sum_i x_{ij}(t+1)$, OPT(conditions sold-out(5) through bang-per-buck(6)) conditions are met.

3. Otherwise, i.e. $\exists i$ s.t. $j \in D_i \bigcap \mathcal{O}_s$ and $r_i \leq \epsilon e_i$

   (a) To ensure that item j is sold out, the item is sold to consumer i s.t. $j \in D_i$. This can only happen if consumer i transfers money from a item that provides lower bang-per-buck, i.e consumer i spends money to purchase item j instead of item k s.t. $x_{ik} > 0 \bigwedge \alpha_{ik} < \alpha_i$(in procedure transfer_money). The procedure transfer_money ensures that sold-out(5) condition is satisfied.

   (b) If item j is still over-supplied, then producers may offer item j at a lower-level price (remember there are 2 price levels). Note that the current price p_j does not change, but consumers will acquire item j more with the same amount of money. When $\sum_s z_{sj}(t+1) - \sum_i x_{ij}(t+1) \leq \epsilon \sum_i h_{ij}(t+1)$, item j will be sold out with the same money(in procedure sell_lprice). Then, $p_j(t) = p_j(t+1)$ and $\sum_i x_{ij}(t) < \sum_i x_{ij}(t+1)$. Conditions OPT will be satisfied.

   (c) Item j will not be sold out yet if $\sum_s z_{sj}(t+1) - \sum_i x_{ij}(t+1) > \epsilon \sum_i h_{ij}(t+1)$. Then, instead of providing item j at lower-level price, producers will offer item j at a lower price than before i.e. $p_j(t+1) = p_j(t)/(1+\epsilon)$(in procedure decrease_price). Then, $p_j(t) > p_j(t+1)$ and $\sum_i x_{ij}(t) < \sum_i x_{ij}(t+1)$. If still $\sum_s z_{sj}(t+1) > \sum_i x_{ij}(t+1)$, then conditions OPT may not be satisfied. However, then the iterations within the procedure bal_os will repeat until $\sum_s z_{sj}(t+1) = \sum_i x_{ij}(t+1)$.

Lemma 5.3 *Condition (5) is satisfied at the end of procedure prod_reschedule.*

Ensuring bpb condition (6):

Lemma 5.4 *Condition (6) is satisfied after procedures satisfy_demand and adjust_bpb.*

Proof: In initialization, $v_{ij}(x_{ij}) \leq \alpha_{ij} p_j$ is true. When procedure adjust_bpb occurs in procedure algorithm_main, $v_{ij}(x_{ij}) \leq \alpha_{ij} p_j$ is always true since adjust_bpb iterates until $\forall j : v_{ij}(x_{ij}) = \alpha_{ij} p_j$. Also, $\alpha_{ij} p_j \leq (1+\epsilon) v_{ij}(x_{ij})$ is true.

In procedure satisfy_demand, note that α_{ij} is updated according to $v_{ij}(x_{ij})$ and p_j which implies that $\forall j : v_{ij}(x_{ij}) = \alpha_{ij} p_j$. For $k = \arg\max_j \alpha_{ij}$, consumer i purchases x'_{ik} amount of item k s.t. $v_{ik}(x'_{ik}) = \alpha_{ik} p_k/(1+\epsilon)$. Then, $v_{ij}(x_{ij}) \leq \alpha_{ij} p_j$ is still true. □

Lemma 5.5 *Condition (6) is satisfied at the end of procedure bal_od.*

Proof: We claim that condition (6) holds at the end of bal_od based on Lemma 5.1.

Let us consider procedure bal_od where consumers balance their over-demand items. Let consumer i give up some amount on item j. Consumer i may potentially violate condition (6) because $v_{ij}(x_{ij})$ increases when x_{ij} decreases.

However, as shown in Lemma 5.1, raise_price occurs at most once at each iteration. Therefore, any price or allocation change does not violate condition (6). □

Lemma 5.6 *Condition bpb (6) is satisfied at the end of procedure bal_os.*

Proof: Let producer s change his production plan, and $j \in \mathcal{O}_s$. There are two possible situations in procedure bal_os: 1)procedure decrease_price is not called. 2)procedure decrease_price is executed.

Procedure decrease_price is not called since consumers consume all amount of item j. Let consumer i have surplus, i.e. $r_i > \epsilon e_i$, and $j \in D_i$. Since consumer i has surplus, she will buy item j as shown in procedure purchase_money. However, it will not violate condition (6) because consumers will only purchase item j within a factor of $(1+\epsilon)$ of bang-per-buck. That is, previously $\alpha_{ij} p_j = v_{ij}(x_{ij})$, and x'_{ij} increases such that $v_{ij}(x_{ij}) \leq v_{ij}(x'_{ij}) \leq (1+\epsilon) v_{ij}(x_{ij})$.

Although consumer i has surplus, the algorithm is allowed for consumers to purchase items by calling procedures transfer_money and sell_lprice. Since the amount of production change is well defined, both procedures do not violate bpb condition. Recall that procedure transfer_money allows consumer i to buy item j to balance bang-per-buck.

When all other procedures do not resolve the over-demand of item j, procedure decrease_price occurs. The occurrence of procedure decrease_price implies that consumers who want to buy item j do not have enough money. After procedure decrease_price occurs, let consumer i buy item j, and let x'_{ij} correspond to the new amount of item j of consumer i.

Previously $\alpha_{ij} p_j = v_{ij}(x_{ij})$ and after one occurrence of procedure decrease_price as shown in Lemma 5.2,

$$(1+\epsilon)\alpha_{ij} = v_{ij}(x_{ij})/p'_j$$

Consumer i now outbids to take item j. Then,

$$\begin{aligned}
&v_{ij}(x_{ij})/(1+\epsilon)p'_j \leq v_{ij}(x'_{ij})/p'_j \leq v_{ij}(x_{ij})/p'_j \\
&(1+\epsilon)\alpha_{ij}/(1+\epsilon) \leq v_{ij}(x'_{ij})/p'_j \leq (1+\epsilon)\alpha_{ij} \\
&\alpha_{ij} \leq v_{ij}(x'_{ij})/p'_j \leq (1+\epsilon)\alpha_{ij}
\end{aligned}$$

□

Ensuring opt-prod condition (7): It happens that no producer reschedule their production plan. When there is no profit for producers, producers do not want to change their own production schedule. Even in the case, we show that producers will satisfy the following inequality as follow:

$$\forall s, j : p_j z_{sj} \leq p_j z^*_{sj} \leq (1+\epsilon)^2 p_j z_{sj}$$

Lemma 5.7 *When the procedure roll_back occurs, producer s still has $O(1+\epsilon)$-approximation optimal profit according to the current prices.*

Proof: Let $V(z,p)$ denote the current profit of producer s on production plan z according to the price vector p. Similarly, let $V(z',p)$ be the next profit of producer s on production plan z' according to the price vector p. $V(\bar{z},p)$ denotes the profit of producer s when he has the production plan $\bar{z}$, the optimal production plan according to the price vector p.

When the production plan shifts from $V(z,p)$ to $V(z',p)$, price may change to p' due to procedure decrease_price. The occurrence of procedure decrease_price may violate that producer s increases his profit. In procedure prod_reschedule, if producer s has no profit increase, then there is no production change. We will show that when procedure roll_back, producer s still guarantees his production profit is well bounded compared with the optimal production profit. We show this by proving that

$$V(\bar{z},p) \leq (1+2\epsilon)V(z,p)$$

Note that $z, z', \bar{z}$ are points on the poly-tope of multiple linear production constraints. By the property,

$$\bar{z} = z + (z'-z)/\sigma \Rightarrow \bar{z}p' = zp' + p'(z'-z)/\sigma$$

where $\sigma = |z'-z|/|\bar{z}-z|$.

$$V(\bar{z},p') = V(z,p') + (V(z',p') - V(z,p'))/\sigma$$

Let $V(z',p') = (1+\epsilon')V(z,p')$,

$$V(\bar{z},p') = V(z,p') + \epsilon' V(z,p')/\sigma$$

Let $\epsilon'/\sigma \leq \epsilon$,

$$V(\bar{z},p') \leq (1+\epsilon)V(z,p)$$

Note that the procedure decrease_price occurs at most once in procedure prod_reschedule which implies that $\forall j : p_j \leq (1+\epsilon)p'_j$.

$$V(\bar{z},p) \leq (1+\epsilon)V(\bar{z},p') \leq (1+2\epsilon)V(z,p)$$

□

6 Analysis of the algorithm

6.1 Invariances

Lemma 6.1 *When the algorithm terminates conditions OPT are satisfied and the algorithm returns a $O(1+\epsilon)$-approximate optimum.*

Proof: Condition sold-out (5) is true at the end of each iteration of the algorithm as shown in Lemma 5.3, condition (5) holds at the end of prod_reschedule.

Condition bang-per-buck (6) is satisfied at the end of algorithm as shown in Lemma 5.4, bpb condition holds after calling procedures adjust_bpb and satisfy_demand. In the case of procedure prod_reschedule, as shown in Lemma 5.4, Lemma 5.5 and Lemma 5.6, condition (6) holds.

Condition (7) is true at the end of prod_reschedule as shown 5.7. □

6.2 Convergence

Now, let us consider time complexity which guarantees that our algorithm converges in We show the convergence of each procedure before we provide time complexity.

Remember that bidding is organized in rounds. In each round every consumer is picked once and reduces his surplus until $r_i = 0$. If there is no outbid, then procedure raise_price will occur followed by procedure prod_reschedule. In procedure prod_reschedule, the algorithm will balance between the demand and the production.

Let $N_0 = \log_{1+\epsilon} \frac{e}{\epsilon e_{min}}$, $N_1 = \log_{1+\epsilon} \frac{p_{max}}{p_{min}} = \log_{1+\epsilon} \frac{e}{\epsilon}$ and $N_2 = \log_{1+\epsilon'} \frac{e}{\epsilon^3 e_{min}}$, where $p_{max} = \max_j p_j$ and $p_{min} = \min_j p_j$.

Claim 6.1 *After $N_1 * N_2$ rounds of bidding, either the algorithm terminates or a round robin completes.*

Proof: If any producer does not reschedule, then price rises. In the worst case, the maximum number of calls to raise_price is bounded by $\log_{1+\epsilon} \frac{p_{max}}{p_{min}}$, termed N_1.

If producers gain profit, then at least one producer will increase her profit by a factor of $(1+\epsilon')$ of the previous profit. We can bound the number of occurrence as $\log_{1+\epsilon'} \frac{e}{\epsilon^3 e_{min}}$, termed N_2.

After $N_1 * N_2$ rounds, procedure algorithm_main will be executed. If there is no consumer with residual money, then the algorithm will exit. Otherwise, the next round robin will occur. □

Let T_{ob}, T_{ls}, T_{bd}, and T_{bs} denote the time taken for procedures outbid, lp_solver, bal_od and bal_os, respectively. $T_{ls} = O(qm^2(m+l)L)$ and $T_{bd} + T_{bs} = mT_{ob} + nm$ where $T_{ob} = \log_{1+\epsilon} \left(\frac{e}{\epsilon}\right)^{|E|} \frac{ev_{max}}{\epsilon v_{min}}$ ($v_{max} = \max_{ij} x_{ij}(0)$ and $v_{min} = \min_{ij} x_{ij}(a_{max})$). Let $|E|$ be the number of nonzero utilities. Also, note that $\epsilon' = \min\{\epsilon^3/e, \max_{ij} v_{ij}(0)/(1+\epsilon) = v((1+\epsilon')\epsilon)$.

Lemma 6.2 *The time complexity of finding equilibrium in MEP is $O(N_0 * N_1 * N_2(T_{ls} + T_{bd} + T_{bs}))$.*

Proof: Let us consider the worst case. For one round-robin, either procedure outbid or raise_price occurs. The number of iterations before a round robin occurs is bounded by $N_1 * N_2$ as shown in Claim 6.1. Note that between round-robins, residual money decreases by a factor of $(1+\epsilon')$ of total money of consumers. It means that total occurrence of round-robins is bounded by $N_0 = \log_{1+\epsilon'} \frac{e}{\epsilon e_{min}}$. Therefore, the total step is bounded by $N_0 * N_1 * N_2$.

Inside procedure prod_reschedule, four events can occur in a call of outbid.

1. y_{kj} becomes zero for some k.
2. r_i becomes zero.
3. α_{ij} reduces by a factor of $(1+\epsilon)$.

4. v_{ij} reaches α_{ij} in the inner while loop of algorithm main.

The number of event (1) is bounded by the number of buyers having non-zero utilities on item j. The total number of (1) events is bounded by $|E| \times N_1$. The number of type (2) events is exactly equal to n in every round of bidding. At every type (3) event, bang-per-buck is reduced by a factor of $(1+\epsilon)$ which varies from $\frac{v_{min}}{p_{max}}$ to $\frac{v_{max}}{p_{min}}$. In every round only one event type of (4) occurs for each buyer.

Thus, outbid takes $|E| \log_{1+\epsilon} \frac{p_{max}}{p_{min}} + \log_{1+\epsilon} \frac{p_{max} v_{max}}{p_{min} v_{min}} = \log_{1+\epsilon} \left(\frac{e}{\epsilon}\right)^{|E|} \frac{e v_{max}}{\epsilon v_{min}}$ as shown in [19].

Note that *lp_solver* is called per each iterations. The function, *lp_solver*, takes $O(qm^2(m+l)L)$, where $m \times l$ represents a matrix of production constraints and L represents an input size. $T_{lp\_solver} = O(qm^2(m+l)L)$. It takes mT_{ob} times for consumers to get money back and for the procedure decrease_procedure, and it takes nm times for producers to sell their produced items. $T_{bd} + T_{bs} = mT_{ob} + nm$.

Time complexity is $O(N_0 * N_1 * N_2(T_{ls} + T_{bd} + T_{bs}))$. □

Theorem 6.1 *Approximation equilibrium in the market equilibrium with production, MEP, can be determined by a PTAS.*

Proof: At termination, termination condition (8) is true because r_i is low. Other conditions are true by Lemma 6.1 which ensures the correctness. The time complexity result follows from Lemma 6.2. □

7 Conclusion

In this paper, we show an auction-based algorithm for a production model where consumers have nonlinear utility functions and producers have a set of linear capacity constraints. Our algorithm can also be extended to arbitrary convex production regions and the Arrow-Debreu model.

References

[1] K. J. Arrow, H. D. Block and L. Hurwicz. On the Stability of the Competitive Equilibrium, *Econometrica*, 27:82-109, 1959.

[2] K. J. Arrow and G. Debreu. Existence of an Equilibrium for a Competitive Economy. *Econometrica*, 22:265-290, 1954.

[3] W. C. Brainard and H. E. Scarf. How to Compute Equilibrium Prices in 1891. *Cowles Foundation Discussion Paper*, 1272.

[4] B. Codenotti and K. Vardarajan. Equilibrium for markets with Leontief Utilies. *ICALP 2004*

[5] B. Codenotti, S. Pemmaraju and K. Varadarajan. A note on the Computation of Equilibria in Exchange Markets with Gross Substitutibility. *Preprint*

[6] B. Codenotti, S. Pemmaraju and K. Varadarajan. On the Polynomial Time Computation of Equilibria for Certain Exchange Economics. *SODA 2005*

[7] B. Codenotti, B. Mccune, S. Pemmaraju, R. Raman and K. Varadarajan. An Experimental Study of Different Approaches to Solve the Market Equilbrium Problem. *ALENEX 2005.*

[8] B. Codenotti, B. Mccune, R. Raman, and K. Varadarajan. Does Theory Meet Practice? *ESA 2005*

[9] B. Codenotti, B. Mccune, and K. Varadarajan. Market Equilibrium via the Excess Demand Function. *STOC 2005*, May 2224, 2005, Baltimore, Maryland, USA.

[10] N. R. Devanur, C. Papadimitriou, A. Saberi, and V. Vazirani. Market Equilibrium via a Primal-Dual-Type Algorithm. *FOCS 2002*, 389-395.

[11] N. R. Devanur, and V. Vazirani. An Improve Approximation Scheme for Computing the Arrow-Debreu Prices for the Linear Case. *FSTTCS 2003*

[12] N. R. Devanur, and V. Vazirani. The Spending Constraint Model for Market Equilibrium: Algorithmic, Existence and Uniqueness Results. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, 2004.

[13] B. Eaves. A Finite Algorithm for the Linear Exchange Model. *Journal of Mathematical Economics*, 3:197-203, 1976.

[14] E. Eisenberg and D. Gale. Consensus of Subjective Probabilities. *The Pari-Mutuel Method, Annals of mathematical Statistics*, 30, 165-168, 1959.

[15] E. Eisenberg. Aggregation of utility functions. *Management Sciences*, 7(4):337-350, 1961.

[16] L. Fleischer, R. Garg, S. Kapoor, R. Khandekar, and A. Saberi. A Fast and Simple Algorithm for Market Equilibria, *WINE 2008*.

[17] R. Garg, and S. Kapoor. Auctin Algorithms for Market Equilibrium. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, 2004.

[18] R. Garg, and S. Kapoor. Auction Algorithms for Market Equilibrium: Price Roll-Backs and Path Auctions for the Fisher Model, *WINE 2007*.

[19] R. Garg, S. Kapoor, and V. Vazirani. An Auction-Based Market Equlilibrium Algorithm for the Separable Gross Substitutability Case, *APPROX 2004*.

[20] G. Goel and V. V. Vazirani. A Perfect Price Discrimination Market Model with Production, and a (Rational) Convex Program for it, *SAGT*, 2010.

[21] K. Jain. A Polynomial Time Algorithm for Computing the Arrow-Debreu Market Equilibrium for Linear Utilities. *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 286-294, 2006.

[22] K. Jain and K. Varadarajan. "Equilibriua for Economies with Productions: Constant-Returns Technologies and Production Planning Constraints", *SODA 2006*.

[23] K. Jain, V. Vazirani, and Y. Ye. Market equilibria for homothetic, quasi-concave utilities and economies of scale in production. *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, January 23-25, 2005.

[24] K. Jain, M. Mahdian, and A. Saberi. Approximating Market Equilibrium. In *Workshop on Approximation Algorithms for Combinatorial Optimization APPROX 2003*.

[25] S. Kapoor, A. Mehta, and V. Vazirani. An Auction-Based Market Equilibrium Algorithm for a Production Model, *Theoretical Computer Science 2007*.

[26] E. I. Nenakov, and M. E. Primak. One algorithm for finding solutions of the Arrow-Debreu model. *Kibernetica*, 3:127-128, 1983.

[27] D. J. Newman and M. E. Primak. Complexity of Circumsribed and Insribed Ellipsoid Methods for Solving Equilibrium Economics Models, *Applied Mathematics and Computation*, 52:223-231, 1992.

[28] M. E. Primak. A converging algorithm for a Linear Exchange Model. *Journal of mathematical Economics*, 22:181-187,1993.

A Appendix - Algorithm

Algorithm A.1 algorithm_main

initialize
while there is extra demand, i.e. $\exists i : r_i > \epsilon e_i$ **do**
  satisfy_demand(i)
  adjust_bpb
  prod_reschedule
end while

Algorithm A.2 initialize

$\forall j : p_j := \epsilon$
$\forall i : \alpha_i := \max_j v_{ij}(0)/p_j$
$\forall i : \mathcal{D}_i := \arg\max_j v_{ij}(0)/p_j$
$\forall s, j : z_{sj} := \epsilon/q$
$\forall i : x_{ij} := \epsilon$, where $j \in \mathcal{D}_i$
ϵ_1 s.t. $\forall i, j : \min_{x_{ij}} v_{ij}(x_{ij})/(1+\epsilon) = v_{ij}((1+\epsilon_1)x_{ij})$
$\epsilon_2 := \epsilon^3/n, n = \sum_i e_i$
$\epsilon' := \min(\epsilon_1, \epsilon_2)$

Algorithm A.3 satisfy_demand(i)

$\forall j : \alpha_{ij} := v_{ij}(x_{ij})/p_j$
$j := \arg\max_l \alpha_{il}$
if $\exists k : y_{kj} > 0$ **then**
  outbid($i, k, j, \alpha_{ij}/(1+\epsilon)$)
  $\alpha_{ij} := v_{ij}(x_{ij})/p_j$
else
  raise_price(j)
end if

Algorithm A.4 adjust_bpb

while $\exists i : r_i > 0$ and $\exists j : \alpha_{ij} p_j < v_{ij}(x_{ij})$ **do**
  if $\exists k : y_{kj} > 0$ **then**
    outbid(i, k, j, α_{ij})
  else
    raise_price(j)
  end if
end while

Algorithm A.5 outbid(i, j, k, α)

$t_1 := y_{kj}$
$t_2 := r_i/p_j$
if $(v_{ij}(a_j) \geq \alpha p_j)$ **then**
  $t_3 := a_j$
else
  $t_3 := \min \delta : v_{ij}(x_{ij} + \delta) = \alpha p_j$
end if
$t := \min(t_1, t_2, t_3)$
$h_{ij} := h_{ij} + t$
$r_i := r_i - t p_j$
$y_{kj} := y_{kj} - t$
$r_k := r_k + t p_j/(1+\epsilon)$

Algorithm A.6 raise_price(j)

$p_j := p_j(1+\epsilon)$
$\forall i : y_{ij} := h_{ij}$
$\forall i : h_{ij} := 0$

Algorithm A.7 prod_reschedule

while *true* **do**
  get the optimal production plan, i.e. $\forall s : \hat{z}_{sj} := lp\_solver(p)$
  $\forall s, j : z'_{sj} := z_{sj} \pm \epsilon' \sum_s z_{sj}/q$
  if $\sum_s \sum_j p_j z'_{sj} > \sum_s \sum_j p_j z_{sj}$ **then**
    bal_od : $\forall j \in \mathcal{O}_d$
    adjust_bpb
    bal_os
    check_profit
  else
    break
  end if
end while

Algorithm A.8 bal_od

while $\exists j \in \mathcal{O}_d^+$ **do**
  if $\exists i : x_{ij} > 0$ **then**
    $t := \min(x_{ij}, \sum_i x_{ij} - \sum_s z_{sj})$
    $x_{ij} := x_{ij} - t$
    $r_i := r_i + tp_j$
  end if
end while

Algorithm A.9 bal_os

while $\forall j \in \mathcal{O}_s$ **do**
  if $\exists i : j \in \mathcal{D}_i$ and $r_i > \epsilon e_i$ **then**
    purchase_money($i, j, \sum_s z_{sj} - \sum_i x_{ij}$)
  else
    if $\exists i : j \in \mathcal{D}_i$ and $r_i \leq \epsilon e_i$ **then**
      transfer_money($i, j, \sum_s z_{sj} - \sum_i x_{ij}$)
    end if
    if producers provide items at lower price, i.e. $\sum_s z_{sj} - \sum_i x_{ij} \leq \epsilon' \sum_i h_{ij}$ **then**
      sell_lprice($i, j, \sum_s z_{sj} - \sum_i x_{ij}$)
    else if Not enough items at lower price, i.e. $\sum_s z_{sj} - \sum_i x_{ij} > \epsilon' \sum_i h_{ij}$ **then**
      decrease_price(j)
    end if
  end if
end while

Algorithm A.10 purchase_money(i, j, t_o)

$t := \min(t_o, r_i/p_j)$
$t_o := t_o - t$
$h_{ij} := h_{ij} + t$
$r_i := r_i - tp_j$

Algorithm A.11 transfer_money(i, j, t_o)

if $t_o > 0$ and $h_{ij'} > 0$ **then**
    $t := \min(h_{ij'} p'_j / p_j, t_o)$
    $h_{ij'} := h_{ij'} - t p_j / p'_j$
    $h_{ij} := h_{ij} + t$
    $t_o := t_o - t$
end if

Algorithm A.12 sell_lprice(i, j, t_o)

while $t_o > 0$ **do**
    if $t_o \geq \epsilon' h_{ij}$ **then**
        $y_{ij} := y_{ij} + (1 + \epsilon') h_{ij}$
        $t_o := t_o - \epsilon' h_{ij}$
        $h_{ij} := 0$
    else
        $y_{ij} := y_{ij} + t_o$
        $h_{ij} := h_{ij} - t_o / (1 + \epsilon')$
        $t_o := 0$
    end if
end while

Algorithm A.13 decrease_price(j)

$\forall i : h_{ij} := (1 + \epsilon) h_{ij} + y_{ij}$
$\forall i : y_{ij} := 0$
$p_j = p_j / (1 + \epsilon)$
$\forall i : \alpha_i := \max_{j'} v_{ij'} / p_{j'}$
$\forall i : \mathcal{D}_i := \arg\max_{j'} v_{ij'} / p_{j'}$

Algorithm A.14 check_profit

let z' and p' be vectors of the previous iteration.
if $\sum_s \sum_j z_{sj} p_j - \sum_j z'_{sj} p'_j < \gamma = \epsilon' \min_s \sum_j p_j z_{sj}$ **then**
    roll_back
    break
end if